De la correction automatisée

C. Queinnec
UPMC - LIP6

Plan

- Le passé
  - DrScheme
  - Notation en masse
- Le futur
  - FW4EX : une infrastructure de correction

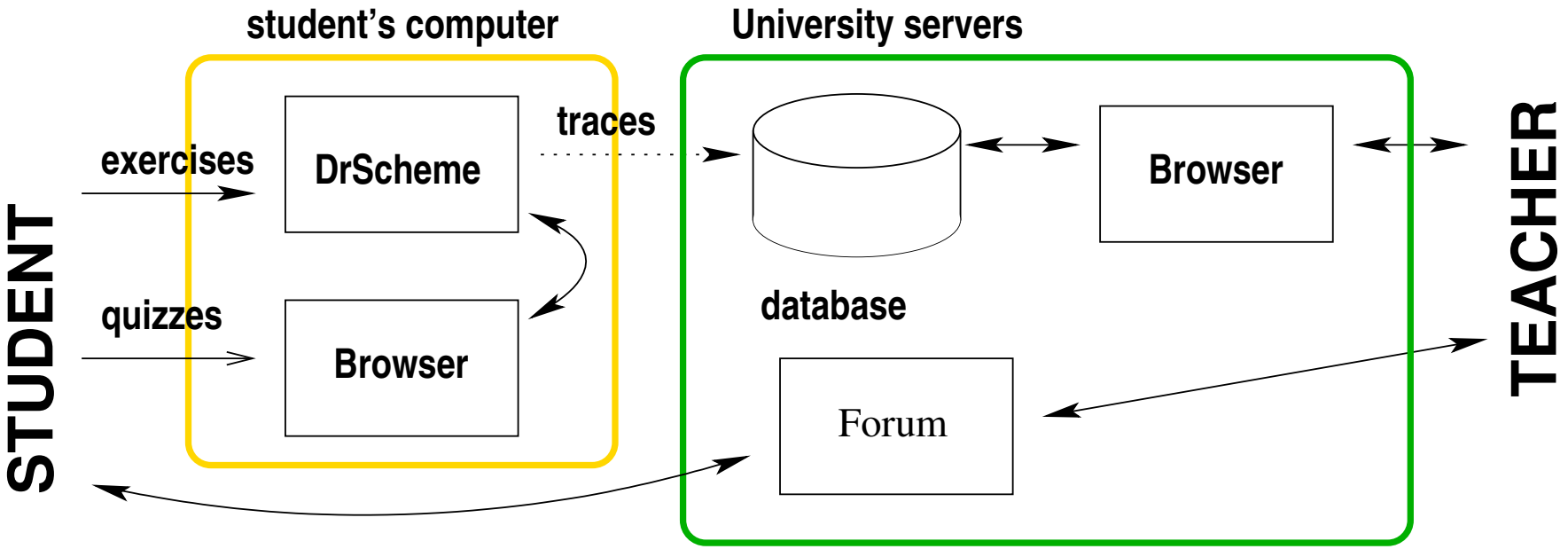
LI101

En 2000, à l'introduction d'un cours « programmation récursive » en premier semestre de L1 : usage de DrScheme avec deux adjonctions

- un serveur web pour des *quizzes*
- un notateur automatique d'*exercices* (couplé avec l'IDE)

pour permettre aux étudiants de pratiquer en tout lieu, à tout instant et de façon autonome (non connecté)

Architecture



Quizzes

Petits exercices en une seule question, automatiquement corrigés, via le web, du genre :

- Quelle est la valeur de ♢
- Donnez une expression ayant pour valeur ♢
- Quel est le type de ♢
- Donnez un contexte où ♢ a pour valeur ♢
- Implanter la spécification ♢

DÉMONSTRATION

Exercices

Buts :

- écrire des fonctions (et leur test)
- programmation plus exigeante avec l'IDE
- automatiquement corrigés à la demande (en local)
- test unitaire boîte noire

Les exercices sont classés, une série progressive est suggérée. Un exercice contient plusieurs questions.

DÉMONSTRATION

Principes de correction d'un exercice

Supposons que l'on demande à l'étudiant une fonction *f*

1. `f` et `(verify f ...)` existent-elles ?
2. `f` satisfait-elle `(verify f ...)` ?
3. la solution de l'enseignant satisfait-elle `(verify f ...)` ?
4. `f` satisfait-elle les tests de l'enseignant ?

L'effet « plus d'une solution ? »
L'effet « assez bon pour continuer »
L'effet « cryptage et logiciel libre »

Détails techniques

- possible grâce à `eval`
- mais nécessite une stricte séparation du code de l'étudiant et de l'enseignant.
- des marges de progression :
  - respect du niveau de langage
  - commentaires situés
  - annotations stylistiques ou algorithmiques
  - vérification des commentaires
  - typage souple

Conclusions sur cette aventure

- exercices souvent utilisés en TP
- les étudiants travaillent à leur vitesse
- les étudiants peuvent travailler chez eux
- taxonomie des erreurs

CFS

Correction en masse

- Entre 2001 et 2004, vérification des facultés de programmation des étudiants de L3
- Grand nombre d'étudiants
- Examen réalisé sur ordinateur

et

- culture du test unitaire
- épreuve (holistique) en condition (sur ordinateur)
- notation uniforme
- publication copies corrigées et notes sur le web
- « annales dynamiques » (E.Chailloux)

Caractéristiques

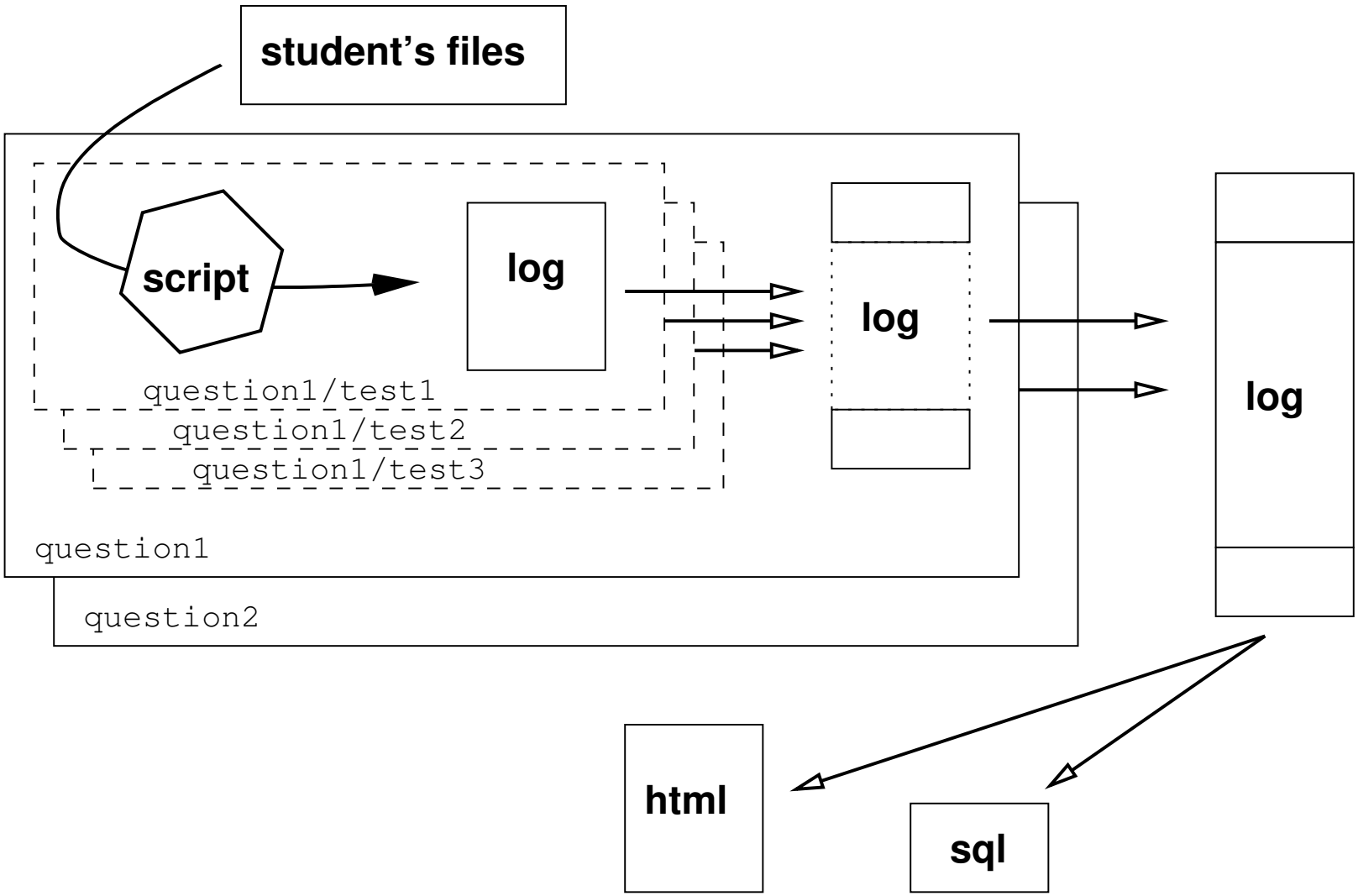
- tests unitaires boîte noire
- spécification ultra-précise du problème
- description des tests pour la notation
- confinement du code
  - programme glouton
  - programme bouclant
  - programme bloqué
  - programme malveillant

Expérimentations

- notation progressive (pas de tout ou rien)
- verbalisation du processus
- pas de question oui-non
- pas d'aléa non maîtrisé
- attention aux dépendances globales (détection de plagiat, compétition, etc.)

- Usage avec bash, make, C, Ada, javascript, Scheme
- en mode client-serveur : perl, php
- avec interfaces et JUnit : java

Cadre conventionnel



Conclusions sur cette aventure

- Techniquement, ça marche !
- Avis mitigé des étudiants
- Investissement conséquent pour les enseignants
  1. énoncé
  2. solution
  3. notateur
- préparation méticuleuse

FW4EX

FW4EX

Vers 2005, proposition d'un *composant* de correction d'exercices

- exercices multi-langages
- exercices facilement déployables
- exercices pérennes

L'imprononçable *FrameWork for EXercises* !

Infrastructure

vers 2007, *infrastructure* de correction d'exercices

- infrastructure robuste, sûre
- toujours disponible
- efficace
- protection vie privée
- topologiquement souple (pare-feux, etc.)
- décorable (*skinnable*)
- passant à l'échelle
- intégrable dans des environnements variés
  - couplée à IDE
  - indépendante des bases de données scolaires
  - indépendante des modes d'authentification

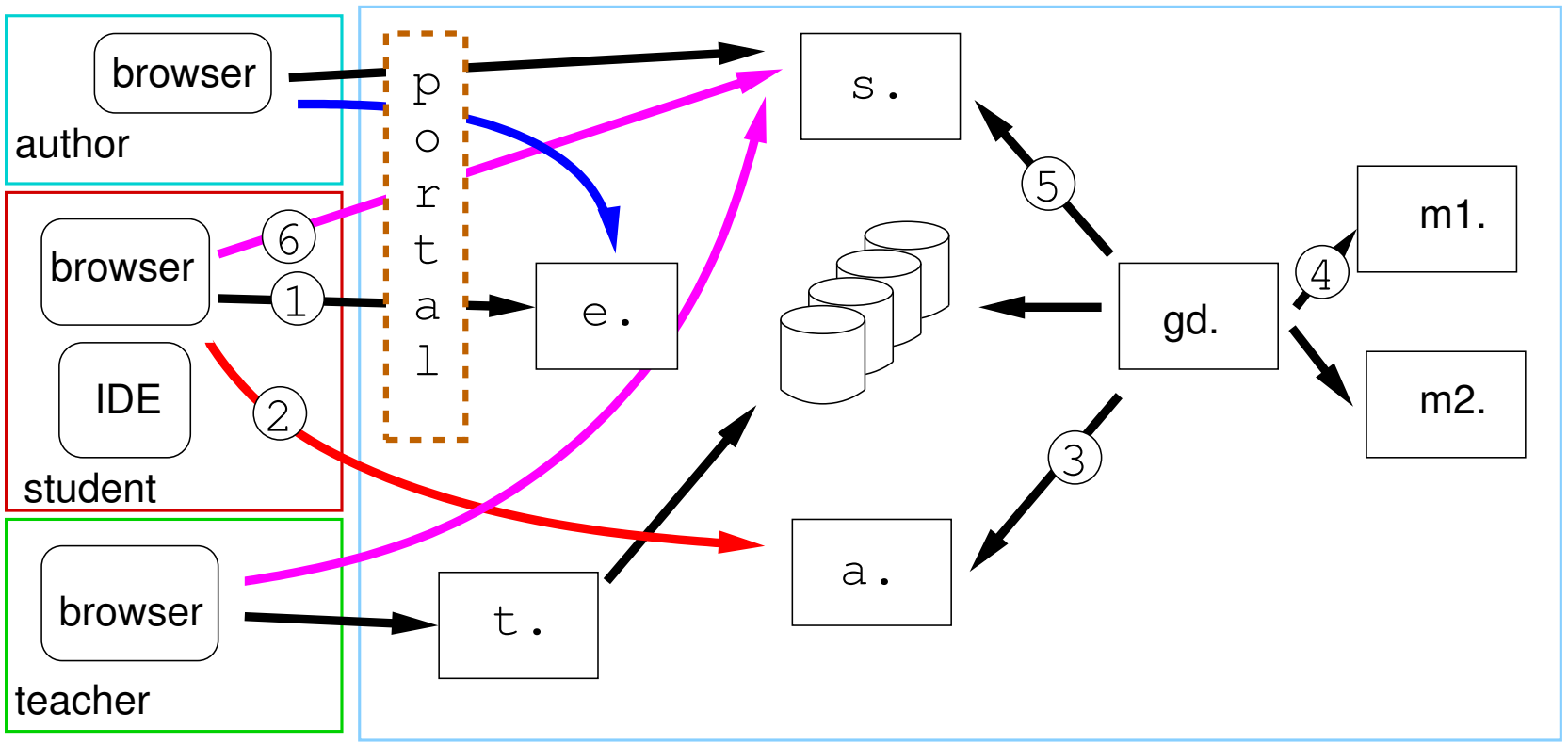
Éco-système

Favoriser l'émergence d'un *éco-système*

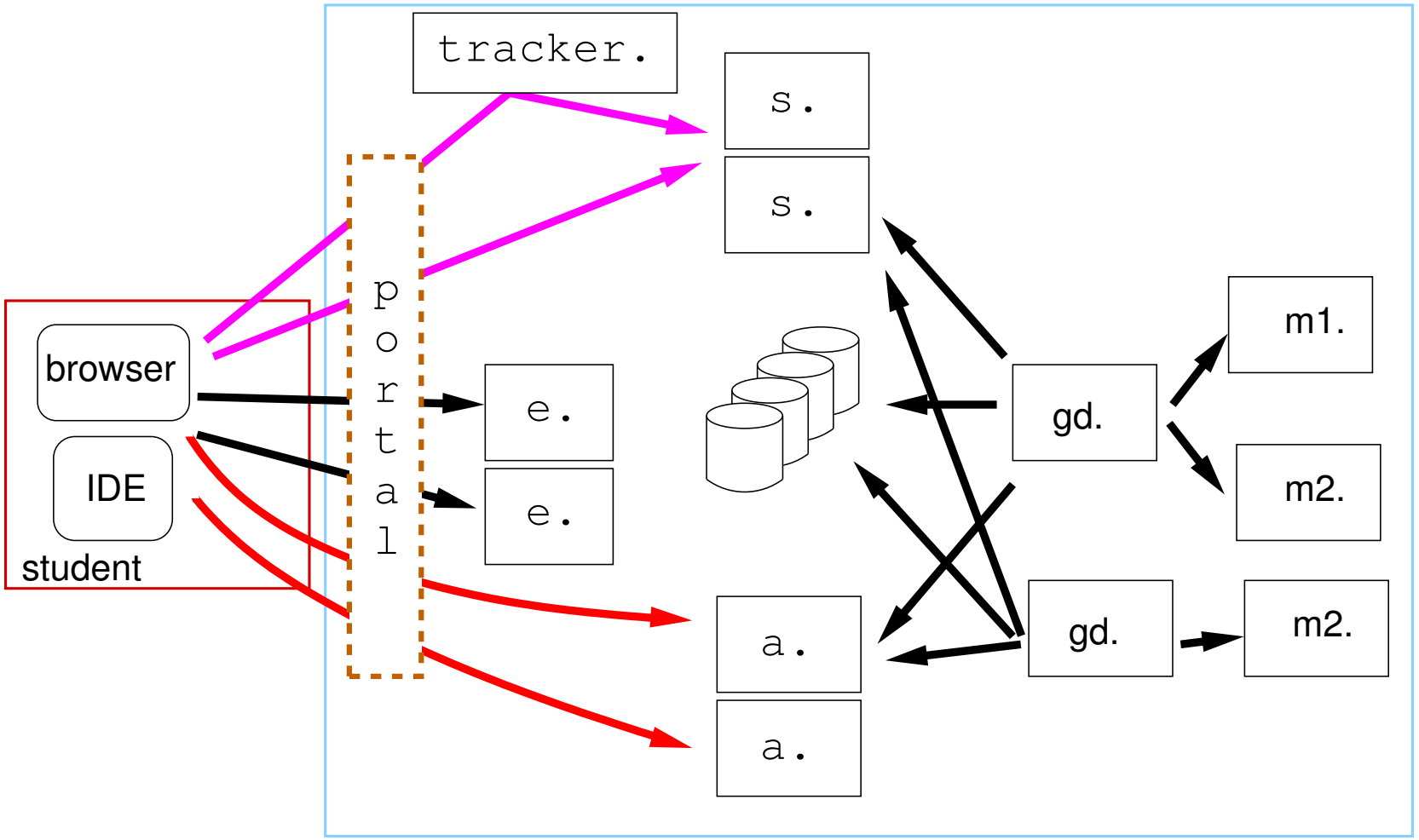
- outils d'écriture pour auteurs d'exercices
- outils d'agencement d'exercices pour enseignant
- outils de déploiement

Architecture

Multiples composants spécialisés activés par des protocoles REST (HTTP) :



Constellation



Base encore à répliquer...

Protocoles

Tous ces serveurs sont actionnables par protocoles (HTTP, REST).
Pour limiter les dépendances aux bibliothèques, OS et aussi pour faciliter le déploiement type EC2 : machines virtuelles indépendantes.
L'usage d'HTTP permet d'envisager des clients légers en Javascript ou en tout langage gérant HTTP.

DÉMONSTRATION

Protocoles REST

- GET http ://proxy.fw4ex.org/path/P
- GET http ://e.fw4ex.org/exercise/E/stem
- POST http ://a.fw4ex.org/exercise/E/job
- GET http ://s.fw4ex.org/J.xml
- GET http ://x.fw4ex.org/jobs
- GET http ://x.fw4ex.org/exercise/E/jobs
- POST http ://e.fw4ex.org/exercises/

Tous les documents échangés sont en XML validés par une grammaire RelaxNG.

Identifiant sûr

Individus, exercices sont représentés par un identifiant non forgeable (un certificat léger) :

Si content = "id=12345, ..."
alors Pub(content ++ Priv(MD5(content)))

Format d'exercice

- un tar gz régi par un descripteur XML
- couvrant tout le cycle de vie de l'exercice

autotest vérifier qu'un exercice est cohérent
publication rendre visible l'exercice
récupération émettre énoncé et fichiers associés vers l'étudiant
installation préparer les fichiers reçus sur la machine de l'étudiant
installation préparer une machine confinée pour noter une soumission
notation noter une soumission de l'étudiant

Écorché d'un exercice

Souvent

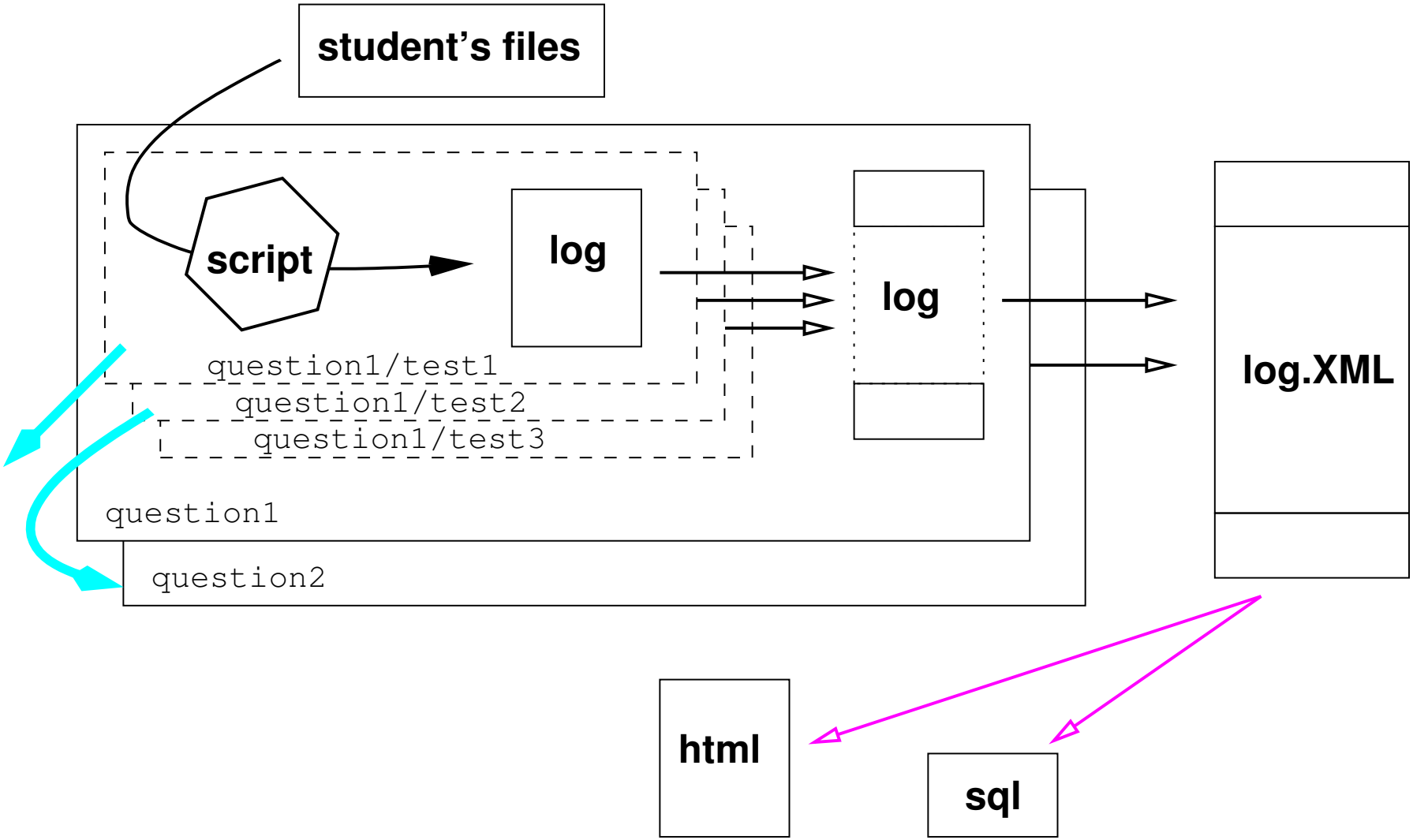
```
./fw4ex.xml
./data/
./scripts/
./pseudos/null/
./pseudos/half/
./pseudos/perfect/
```

Squelette d'un notateur de question

1. Vérification des fichiers attendus
2. Affichage des fichiers soumis
3. Boucle sur les tests
   - 3.1 Préparation
   - 3.2 Évaluer réponse étudiant
   - 3.3 Montrer résultat étudiant
   - 3.4 Normaliser résultat étudiant
   - 3.5 Appréciation résultat (ici par comparaison)
     - 3.5.1 Évaluer solution auteur
     - 3.5.2 Montrer résultat auteur
     - 3.5.3 Normaliser résultat auteur
   - 3.6 Détermination du gain

Flux

Les fichiers de l'auteur sont déployés, dans l'esclave de notation (une machine virtuelle), dans le HOME de l'auteur.
Les fichiers de l'étudiant sont déployés, dans l'esclave de notation (une machine virtuelle), dans le HOME de l'étudiant.
Les scripts de l'auteur de l'exercice y sont exécutés selon un enchaineur de scripts compilé depuis le descripteur XML.
Un script tourne dans le HOME de l'étudiant, produit un fragment de rapport sur son flux de sortie (incluant les notes partielles) et retourne un code d'erreur. Le flux d'erreur du script est pour l'auteur.


student's files
script
log
log
log.XML
question1/test1
question1/test2
question1/test3
question1
question2
html
sql

Scripts

depuis

```
for test in ...
do
  fw4ex_confine ... studentprog ... \
      >$TMPDIR/out.s 2>$TMPDIR/err.s
  fw4ex_verbalize_exit_code
  fw4ex_verbalize_stderr
  # analyse out.s
  exit $(cat $TMPDIR/.lastExitCode)
done
```

jusqu'à (quand les bibliothèques sont au point) :

```
source bibliotheque.sh
WIN_FORMULA='triangular 5 10 12 23'
go
```

Trois fois plus de travail !

- écrire énoncé
- écrire solution (réviser énoncé)
- écrire notateur (réviser énoncé et solution)

Et, plus tard, analyse des erreurs...

Expériences

one liner suite d'exercices (utilitaires d'Unix) : les options utiles de `tr`, `sort`, `grep`, etc.

- sommairement spécifié
- testé avec des cas limites
- incitation à lire les pages de manuel

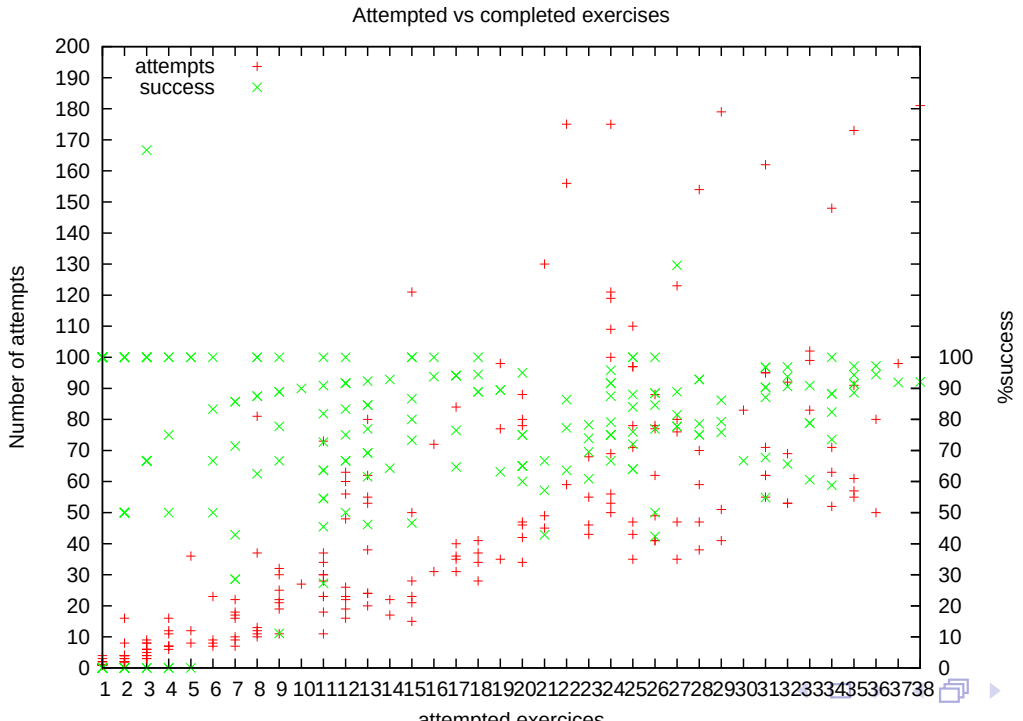
examen rejouer un précédent examen

JFP journées franciliennes de programmation

Exercices en sh, make, Java, C, Octave, Perl.

Retour d'expériences

- 10 000 corrections depuis septembre 2008
- entre 30 et 200 étudiants concernés par exercice
- entre 1 et 51 essais par exercice (en moyenne 3)
- entre 25% et 98% (moyenne 75%) des étudiants finissent par réussir
- rejeu d'examens peu exploité



Contributions

sur SourceForge (licence BSD).

- en 2009 :
  - client pour C et SCiTE (Yohan Bittan, Cai Bo)
  - client pour Java et Eclipse (Tina Alaeitabar, Yassamine Seladji)
  - greffon pour auteur en Eclipse (Frédéric Ye, Mickael Zhu)
- en cours :
  - client pour Java et Eclipse (Maxime Claudel, Edwige Zohi)
  - génération de scripts de notation en Eclipse (Christophe Bigot, Gabriel Szekely)

Modèle économique

Est-ce viable ?

- L'auteur choisit le prix de vente et le temps maximal de correction
- l'infrastructure retient le prix correspondant au temps maximal de correction (avec une petite marge)
- Le reste va à l'auteur.

Aujourd'hui, 1 centime d'€ équivaut à environ 3 minutes de correction.

Fin ?

Travaux futurs

- Amélioration des outils pour auteurs
- Accroître l'émulation (Twitter, etc.)
- Nouveau client (la regexp underground)
- Taxonomie des erreurs

Ultimes conclusions

- La notation automatisée existe, progresse, s'étend
- Infrastructure ouverte (gratuitement) pour vos essais
- et, en prime, sinon la richesse, l'immortalité pour vos exercices !